T-122.102 Special Course in Information Science VI:
Co-occurence methods in analysis of discrete data

Kernels for Structured Data

Based on article: *A Survey of Kernels for Structured Data*
by Thomas Gärtner

Pasi Lehtimäki
Pasi.Lehtimaki@hut.fi
46478E

February 9, 2004

1 Introduction

Kernel methods have become popular in supervised learning tasks, but also in unsupervised learning. The main ingredient of kernel methods is the use of a separate feature space when solving the problem at hand. For example, a kernel method for implementing a nonlinear classifier is typically obtained by mapping the data from the input space into a feature space via a nonlinear mapping, and solving the classification problem in the feature space. After the data is mapped into the feature space, a simple classifier or function is fitted to the data. The most attractive feature of kernel methods is that the fact that the mapping into the feature space need not be stated explicitly. In addition, a global solution for the optimization problem can be obtained.

1.1 The kernel trick

The most important issue in kernel methods is the fact that the mapping from input space to the into the feature space need not be explicit. This can be seen by selecting a mapping Φ from input space to feature space as:

$$\Phi : (x_1, x_2) \rightarrow (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

It is straightforward to see that this corresponds to kernel $K(x, x') = \langle x, x' \rangle^2$:

$$\begin{aligned} K(x, x') &= \langle x, x' \rangle^2 = (x_1x_1' + x_2x_2')^2 \\ &= (x_1x_1')^2 + 2x_1x_2x_1'x_2' + (x_2x_2')^2 \\ &= \langle \left(x_1^2, \sqrt{2}x_1x_2, x_2^2 \right), \left(x_1'^2, \sqrt{2}x_1'x_2', x_2'^2 \right) \rangle \\ &= \langle \Phi(x), \Phi(x') \rangle. \end{aligned}$$

Therefore, the mapping Φ is not explicitly needed. Instead, one can evaluate the value of the kernel function $K(x, x')$.

1.2 Valid and good kernels

Kernel $K(x, x')$ measures the similarity of x and x'. The requirement for a function $K(x, x')$ to be a valid kernel is that it must be positive definite. However, there is always a valid kernel performing poorly at a problem, but also there is always a kernel performing ideally. Three distinct characteristics for a good kernel have been proposed: completeness, correctness and appropriateness.

Completeness refers to the extent to which the knowledge incorporated to the kernel is sufficient for solving the problem at hand. *Correctness* refers to the extent to which the underlying semantics of the problem are obeyed in the kernel. *Appropriateness* refers to the extent to which the examples that are close to each other in the class membership are also close to each other in the feature space.

2 Model-driven kernels

There are various approaches for deriving a suitable kernel for the problem at hand. *Model-driven* kernels are generated from the knowledge about the semantics of the problem domain or estimated from data. *Syntax-driven* kernels are derived from the semantics of the data and are described in the next section.

2.1 Kernels from generative models

Let us assume that we have a set of possible states $s_i, i = 1, ..., N$, and a state transition matrix A of size $N \times N$, where A_{ij} captures the probability of the system to switch from state s_i to s_j. A generative model for the system is of the form $P(s|\theta)$, where θ consists of the probabilities in the matrix A. Let U_x be the gradient of the log-likelihood w.r.t the parameters of the generative model $P(x|\theta)$ at x:

$$U_x = \nabla_\theta \log P(x|\theta).$$

Now, a kernel that can be used to process sequences that are produced by a generative model $P(x|\theta)$ is

$$K(x, x') = U_x^T U_x.$$

Often, the kernel is equipped with the Fisher information matrix I over the distribution $P(x|\theta)$:

$$I = E\{U_x U_x^T\},$$

yielding into so-called Fisher kernel:

$$K(x, x') = U_x^T I^{-1} U_x.$$

2.2 Kernel from transformations

The so-called diffusion kernel is obtained through a transformation using matrix exponentiation. It can be presented as

$$K = e^{\beta H} = \lim_{n \to \infty} \left(1 + \frac{\beta H}{n}\right)^n,$$

where β is so-called bandwidth parameter and H is a *generator*. Differentiating with respect to β leads into differential equation

$$\frac{d}{d\beta} K(\beta) = H K(\beta).$$

Selecting initial conditions $K(0) = I$ leads into interpretation that $K(\beta)$ is the product of continuous process, expressed by H, gradually transforming it from identity matrix $K(0)$ to a kernel with stronger and stronger off-diagonal effects as β increases. Thus, choosing H to express the *local* structure of input data will result in the *global* structure of the input data naturally emerging in K.

For example, an undirected graph G is defined by a vertex set V and an edge set E, where $\{v_i, v_j\} \in E$ if there is an edge between vertices v_i and v_j. A suitable generator is

$$H_{ij} = \begin{cases} 1 & ,\{v_i, v_j\} \in E \\ -d_i & ,i = j \\ 0 & ,\text{otherwise} \end{cases}$$

where d_i is the number of edges originating from vertex v_i.

3 Syntax-driven kernels

3.1 Convolution kernels

The semantics of the composite objects can often be captured by a relation R between the object and its parts. Let $\vec{x} = x_1, x_2, \ldots, x_d$ denote the parts of object x, and R be a relation on the set $X_1 \times X_2 \times \ldots \times X_d \times X$. $R(\vec{x}, x)$ is true iff $x_1, x_2, \ldots, x_d$ are the parts of x. Let $R^{-1}(x) = \{\vec{x} : R(\vec{x}, x)\}$ be the set of parts of x. Let us assume, that kernel $K_d(x_d, x'_d)$ measures the similarity of part d of the objects x and x'. Then, a convolution kernel suitable for measuring the similarity of composite objects x and x' is:

$$K(x, x') = \sum_{\vec{x} \in R^{-1}(x), \vec{x}' \in R^{-1}(x')} \prod_{d=1}^{D} K_d(x_d, x'_d)$$

3.2 String kernels

The development of methods to process character strings is of high importance in many areas. Using kernel methods, the similarity of two strings s_1 and s_2 (and thus, the value of the kernel function $K(s_1, s_2)$) can be based on the number of common subsequences and by penalizing the occurences of g aps within the subsequences.

For example, consider two strings s_1="cat" and s_2="cart". The common subsequences occurring in both sequences are “c”, “a”, “t”, “ca”, “at”,“ct”, “cat”. Now, the total length of occurences of these subsequences in s_1 and s_2 are (w.r.t s_1, w.r.t s_2): “c”(1,1), “a”(1,1), “t”(1,1), “ca”(2,2), “at”(2,3), “ct”(3,4), “cat”(3,4).

Now, using a decay factor λ, penalties corresponding to the subsequences become “c”: $\lambda^1\lambda^1$, “a”:$\lambda^1\lambda^1$, “t”:$\lambda^1\lambda^1$, “ca”:$\lambda^2\lambda^2$, “at”:$\lambda^2\lambda^3$, “ct”:$\lambda^3\lambda^4$, “cat”:$\lambda^3\lambda^4$. Now, the value of the kernel function between two strings s_1 and s_2 is the sum of the penalties:

$$K(\text{“cat”}, \text{“cart”}) = 2\lambda^7 + \lambda^5 + \lambda^4 + 3\lambda^2$$

However, the computation of the value of this kind of kernels may be very expensive.

3.3 Tree kernels

Let us assume, that the instances considered in the learning task are labeled and ordered directed subtrees. Now, consider some enumeration of all possible subtrees and let $h_i(T)$ be the number of occurences of ith subtree in tree T. In order to measure the similarity of two trees T_1 and T_2, the value of the kernel

$$K(T_1, T_2) = \sum_i h_i(T_1) h_i(T_2)$$

can be computed, giving an intuitive measure between the similarity of the graphs T_1 and T_2.

3.4 Basic term kernels

The key idea is to use fixed type structures. In this context, three kind of types are used: function types, product types and constructor types. Function types are used to represent objects corresponding to sets and multisets. Product types represent objects corresponding to tuples and constructors are used to represent structural objects of arbitrary size, such as lists, trees etc.

Each type defines *basic terms* representing the instances of the types. *Abstraction* is used to build instances of function type, *tupling* is used to create instances of product type and *application* corresponds to building objects of a type constructor.

For example, basic term s represents the set $\{1, 2\}$ and basic term t represents the multiset with 42 occurences of A and 21 occurences of B:

$$\begin{aligned} s &= \lambda x.\text{if } x = 1 \text{ then } \top \text{ else if } x = 2 \text{ then } \top \text{ else } \bot \\ t &= \lambda x.\text{if } x = A \text{ then } 42 \text{ else if } x = B \text{ then } 21 \text{ else } 0 \end{aligned}$$

For basic abstraction r, $V(r\ u)$ denotes the value of r when applied to u. For example, $V(s\ 2) = \top$ and $V(t\ C) = 0$. *Support* of an abstraction is the set of terms u for which $V(r\ u)$ differs from default value. For example, $supp(s) = \{1, 2\}$. Now, if s and t are basic terms formed by abstraction, then a suitable kernel to measure their similarity is:

$$K(s, t) = \sum_{u \in supp(s), v \in supp(t)} K(V(s\ u), V(t\ v)) \cdot K(u, v).$$

3.5 Graph kernels

A graph consist of a set of vertices, a set of edges between the vertices, a set of labels for the vertices and a set of labels for the edges. Two graphs generating a product graph are called factor graphs. The vertex set of the product graph is Cartesian product of the vertex sets of the factor graphs. The product graph has a vertex iff the labels of the corresponding vertices in the factor graphs are the same. There is an edge between two vertices in the product graph if there

is an edge between the corresponding vertices in both factor graphs and both edges have the same label.

Let's denote the edge set of the product graph by $\mathcal{E}_\times$. Let's denote an enumeration of vertex set by $\mathcal{V} = \{v_i\}$, $i = 1, ..., N$. The elemenents of the so-called adjacency matrix $E_\times$ are defined by $[E_\times]_{ij} = 1 \iff (v_i, v_j) \in \mathcal{E}_\times$, and $[E_\times]_{ij} = 0 \iff (v_i, v_j) \notin \mathcal{E}_\times$. With a sequence of weights $\lambda_0, \lambda_1, \ldots,$ $(\lambda_i \geq 0, \forall i)$, the value of the product kernel between two graphs G_1 and G_2 is:

$$K_\times(G_1, G_2) = \sum_{i,j=1}^{|\mathcal{V}_\times|} \left[\sum_{n=0}^{\infty} \lambda_n E_\times^n \right]_{ij},$$

where $\left[E_\times^n\right]_{ij}$ is the number of walks of length n from v_i to v_j.

4 Conclusions

Kernel methods, especially for structured data is a promising research area. For discrete data, the problem reduces in selecting a suitable kernel. In the literature, *a lot* of kernels have been proposed. The selection of a suitable kernel for the problem at hand is not a trivial task. A lot of applications for processing sequences describing DNA, protein, gene, speech, text, molecule, etc. are presented.